Project 2

Foundation of robotics

Shantanu Kumar
Sk9698

NYU

Question 1: inverse kinematics

a)

Write a function `compute_IK_position` that gets a desired end-effector 3D position (in spatial frame) and returns a vector of joint angles that solves the inverse kinematics problem

I have written a function for computing inverse kinematics through the following steps:

- I used forward kinematics to find the Homogeneous matrix of the body in spatial frame.
- I calculate Jacobian using the initial theta values.
- I orient the Jacobian to a frame present at the end-effector position but oriented as the spatial frame.

```
def get_spaceoribody_jacobian(T, J):
  r = np.eye(3)
  p = -T[0:3,3]
  A_up = np.concatenate((np.eye(3),np.zeros((3,3),int)), axis=1)
  A_down = np.concatenate(((vec_to_skew(p) @ r),r), axis=1)
  a=np.concatenate((A_u,A_d), axis=0)
  return a @ J
```

- I take the last three values which represent linear velocity, and take their pseudo inverse using the formulae:

  ```
  J_sudo = (np.transpose(J) @ inverse(J @ np.transpose(J) + epsilon*np.eye(3)))
  ```

  Where epsilon is the regularization term with a value of 10^-4 to 10^-6

- Calculate error as the difference in current position and the desired position.
- Multiply the error by the pseudo inverse to get the desired theta.
- Changing the current position would change the Jacobian and therefore our desired theta will change, but if we do them in small intervals overtime, we can reach the desired position.

- The min values of error are set so that the loop breaks when the required precision is hit.
- There can be infinite solutions to each position as our arm has 7 degrees of freedom, the specific solution to our given position would depend upon the alpha and the starting configuration of the robot.
- Taking an higher or lower alpha value can result in over or undershoot.

b)

The file `desired_end_effector_positions.npy` contains a sequence of 10 desired end-effector positions. For all the positions attainable by the robot, compute an inverse kinematics solution. For the positions for which an inverse kinematics solution does not exist, what is the issue and how close can you get the end-effector to the desired position?

I used the inverse kinematics function to find the following theta values:

```
[array([-1.21216215,  1.978781  , -0.25966358,  1.7397395 ,  0.97031557,
       -0.77441971, -0.2271579]),
array([ 0.78143573,  1.60974559,  2.68832224, -1.25539094,  2.44340195,
        0.35794272, -0.47904396]),
array([ 2.02415495,  0.4447174 ,  0.6324689 , -1.36716257,  2.42611928,
       -2.89514374, -0.65346482]),
array([ 2.01668958, -0.71665893,  0.36855513, -1.24838011, -0.44661505,
            -0.78105049, -0.1725567]),
array([-0.05205253,  0.60405866,  1.86572114,  1.6381175 ,  0.697895  ,
1.88632915, -0.48089357]),
array([ 2.04656528e+00,  3.34986723e-01,  1.65316498e+00, -3.63331015e-04,
        2.58405838e+00, -4.50771176e-03, -8.47551047e-02]),
array([-0.47299276, -0.73554102, -1.37397249,  0.04942642,  2.39874936,
            -0.14769304, 0.27845298]),
array([-1.95563915,  1.26000525, -2.32736241, -0.32619621, -2.23313791,
       -0.25120674, -2.03541193]),
array([ 1.62183583,  1.09315459,  0.18250697,  1.97352434,  1.35558979,
        0.35734971, -0.38859997]),
array([ 2.59327262, -0.71272889,  0.30495297,  0.87858222,  1.45420549,
       -1.04772753, -0.5226792])]
```

Error Values:

```
0.6814517003574615,
0.0006119418845539743,
0.0009223451558902806,
0.0005319913571550034,
0.0002665650546855888,
0.07145187489936562,
0.36536610710340395,
0.06155207541679003,
```

```
0.0007243628793600945,
0.000729896189616459
```

- The results were evaluated for alpha = 0.02 and 10000 iterations.
- 6 out the 10 values converged with an error of <0.001
- 2 partially converged with an error of about 0.06-0.07(6^{th} and 8^{th})
- 2 did not converge with error of 0.68 and 0.36(1^{st} and 7^{th})

I think that the position 1^{st} and 7^{th} might be out of reach for the robot and the position 6^{th} and 8^{th} might be present at a singularity or they are also out of reach. So, out robot is not able to reach the spots. The robot can reach the desired point with an accuracy of 0.001. If the base of the robot can move it would be able to reach any position.

c)

Write a function `compute_IK_position_nullspace` that solves the inverse kinematics problem and additionally uses joint redundancy (i.e. the nullspace) to try and keep the joints close to the following configuration $[1,1,-1,-1,1,1,1]$[1,1,−1,−1,1,1,1]. Explain how you used the nullspace to implement this function.

- We know that IK with redundancy can be given by

$$\Delta\theta = J^{+}.\Delta x + (I - J^{+}J)\bar{\theta}$$

- I have implemented the above given equation in my code where $\bar{\theta}$ indicated the difference in desired positions.
- I know the term$(I - J^{+}J)\bar{\theta}$ projects orthogonal projections in the null space.

Implementing the formula, I ran Inverse kinematics with null space for 10000 iterations with alpha value qual to 0.02. The results were as follows:

Theta:

```
[array([-1.21216215,  1.978781  , -0.25966358,  1.7397395 ,  0.97031557,
       -0.77441971, -0.2271579 ]),
array([ 0.78143573,  1.60974559,  2.68832224, -1.25539094,  2.44340195,
```

```
        0.35794272, -0.47904396]),
array([ 2.02415495,  0.4447174 ,  0.6324689 , -1.36716257,  2.42611928,
       -2.89514374, -0.65346482]),
array([ 2.01668958, -0.71665893,  0.36855513, -1.24838011, -0.44661505,
       -0.78105049, -0.1725567 ]),
array([-0.05205253,  0.60405866,  1.86572114,  1.6381175 ,  0.697895  ,
        1.88632915, -0.48089357]),
array([ 2.04656528e+00,  3.34986723e-01,  1.65316498e+00, -3.63331015e-04,
        2.58405838e+00, -4.50771176e-03, -8.47551047e-02]),
array([-0.47299276, -0.73554102, -1.37397249,  0.04942642,  2.39874936,
       -0.14769304,  0.27845298]),
array([-1.95563915,  1.26000525, -2.32736241, -0.32619621, -2.23313791,
       -0.25120674, -2.03541193]),
array([ 1.62183583,  1.09315459,  0.18250697,  1.97352434,  1.35558979,
        0.35734971, -0.38859997]),
array([ 2.59327262, -0.71272889,  0.30495297,  0.87858222,  1.45420549,
       -1.04772753, -0.5226792 ])]
```

Error

```
0.6814517003574615,
0.0006119418845539743,
0.0009223451558902806,
0.0005319913571550034,
0.0002665650546855888,
0.07145187489936562,
0.36536610710340395,
0.06155207541679003,
0.0007243628793600945,
0.000729896189616459
```

We can see that using nullspace, gave us theta values to a reach a specific point in space while trying to maintain a specific position for the robot's joints. It will reduce the infinite number of solutions present in the range of the robot to a lower number, in some cases a unique solution.

d)

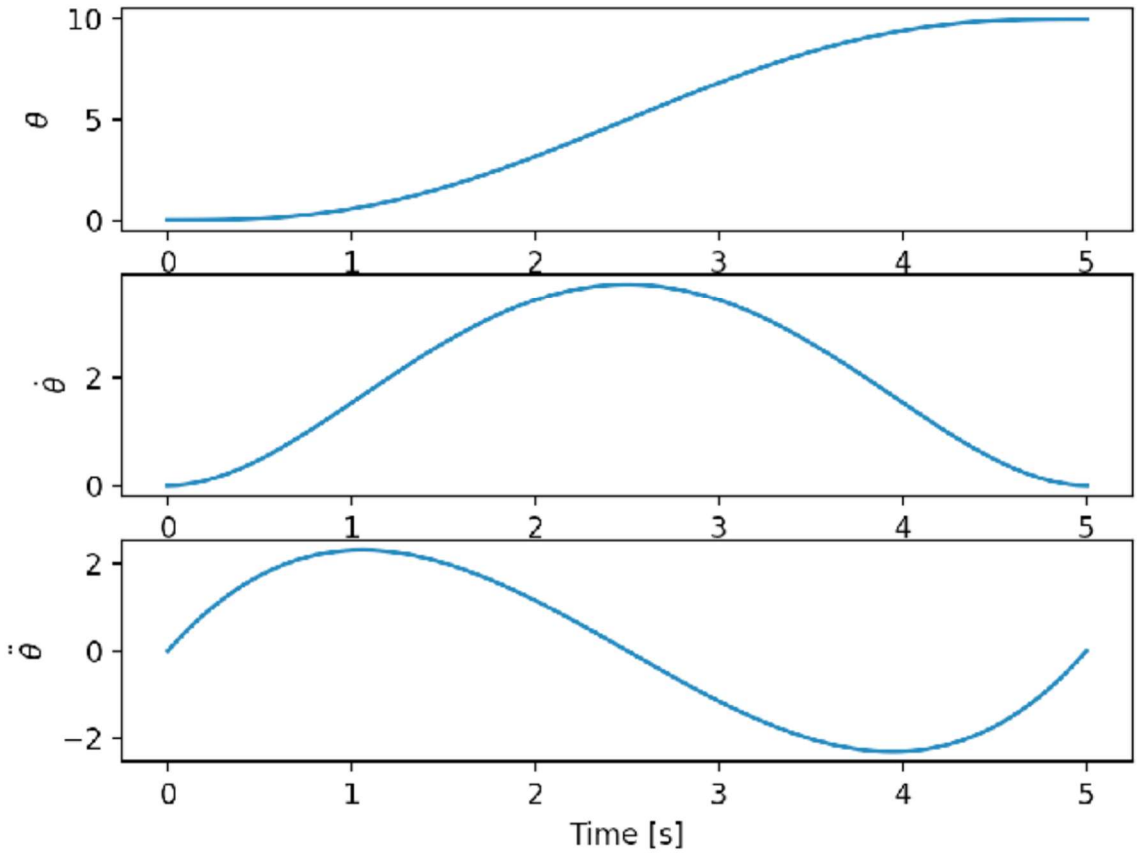
Use this new function to reach the positions set in the file `desired_end_effector_positions.npy`, how do the solutions compare to the first ones you found?

- The results were evaluated for alpha = 0.02 and 10000 iterations.
- 6 out the 10 values converged with an error of <0.001
- 2 partially converged with an error of about 0.06-0.07(6^{th} and 8^{th})
- 2 did not converge with error of 0.68 and 0.36(1^{st} and 7^{th}

Question 2: Joint control and joint trajectories generation

- Formula for desired trajectory:

$$\theta_{des}(t) = \theta_{init} + \left(\frac{10}{T^3}t^3 + \frac{-15}{T^4}t^4 + \frac{6}{T^5}t^5\right)(\theta_{goal} - \theta_{init})$$

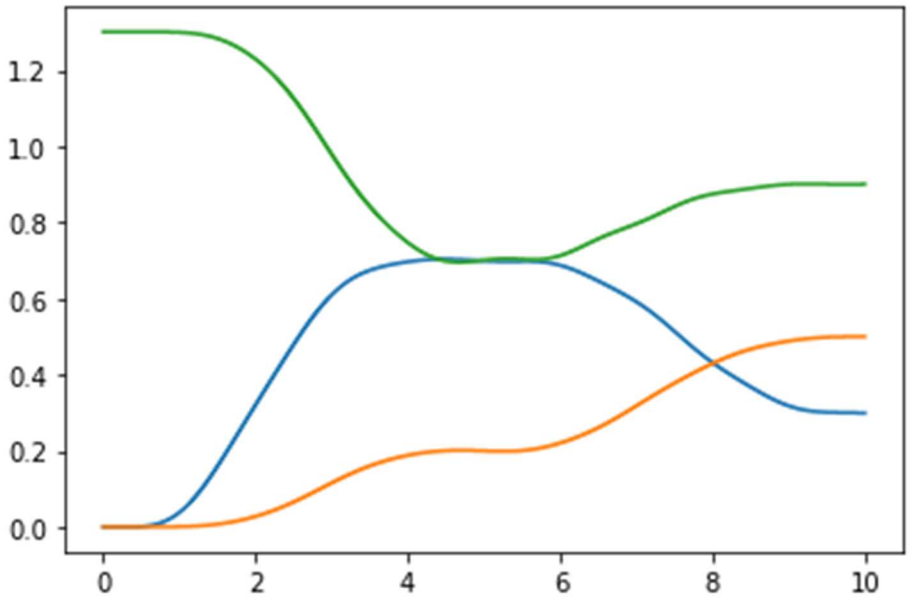


We use this equation to write a function which returns the desired velocity and position so that the acceleration and velocity are zero at the start and finish of our simulation.

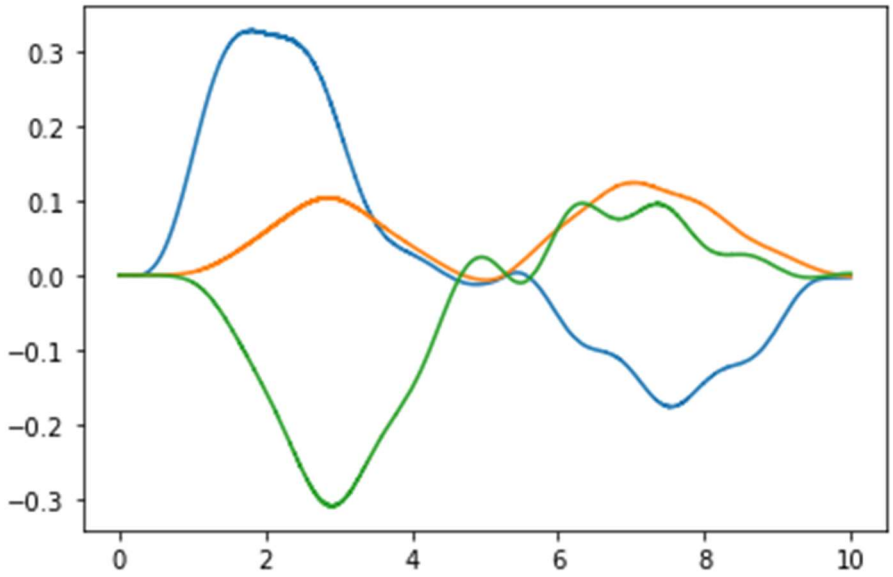
```
def get_point_to_point_motion(T, initial_p, final_p):

  def theta(t):

    return initial_p + ((10*(t**3)/T**3 - 15*(t**4)/T**4 + 6*(t**5)/T**5) * (final_p - initial_p))
```

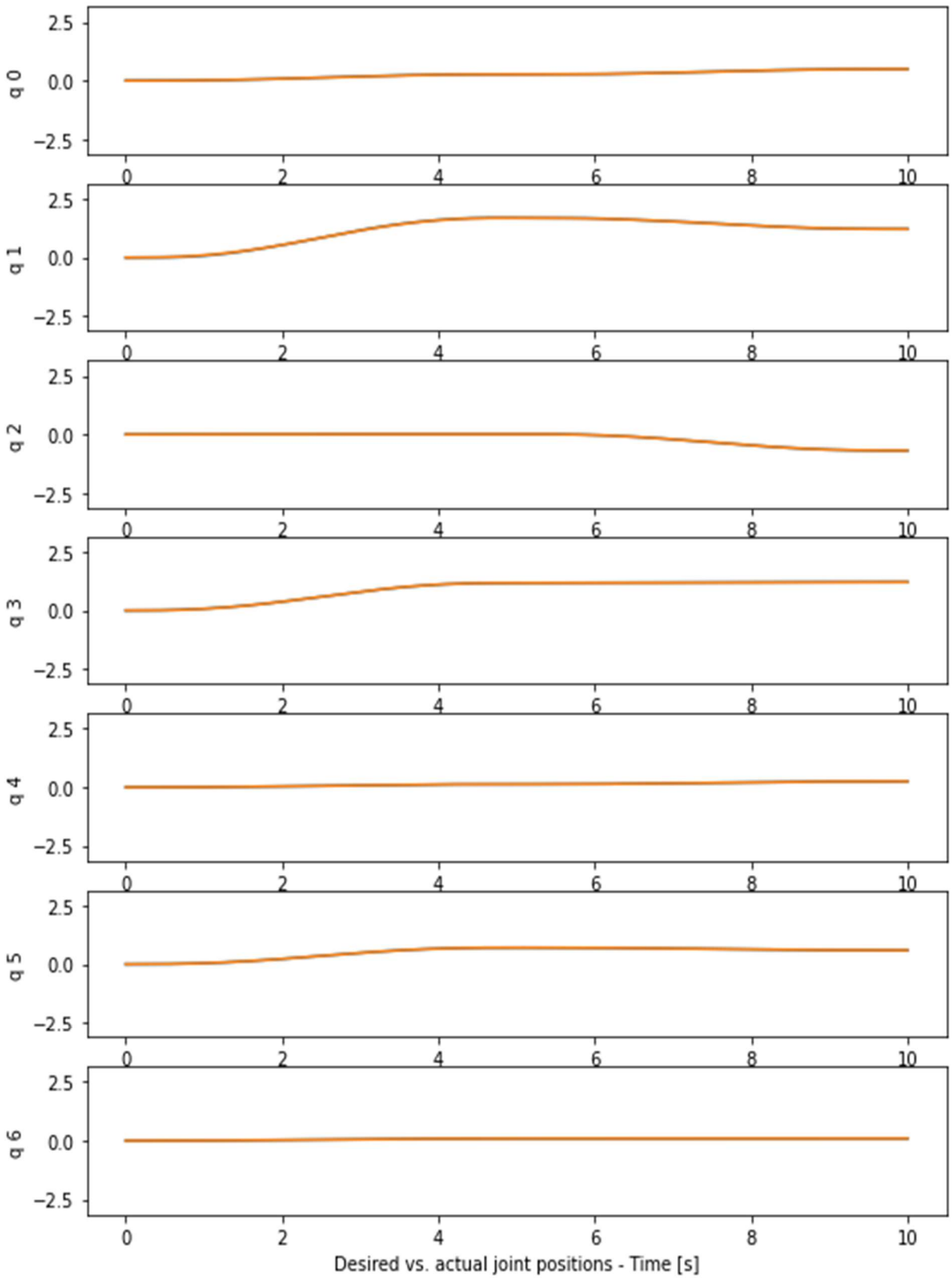
```
def dtheta(t):
  return (30*(t**2)/T**3 - 60*(t**3)/T**4 + 30*(t**4)/T**5) * (final_p - initial_p)
return theta, dtheta
```

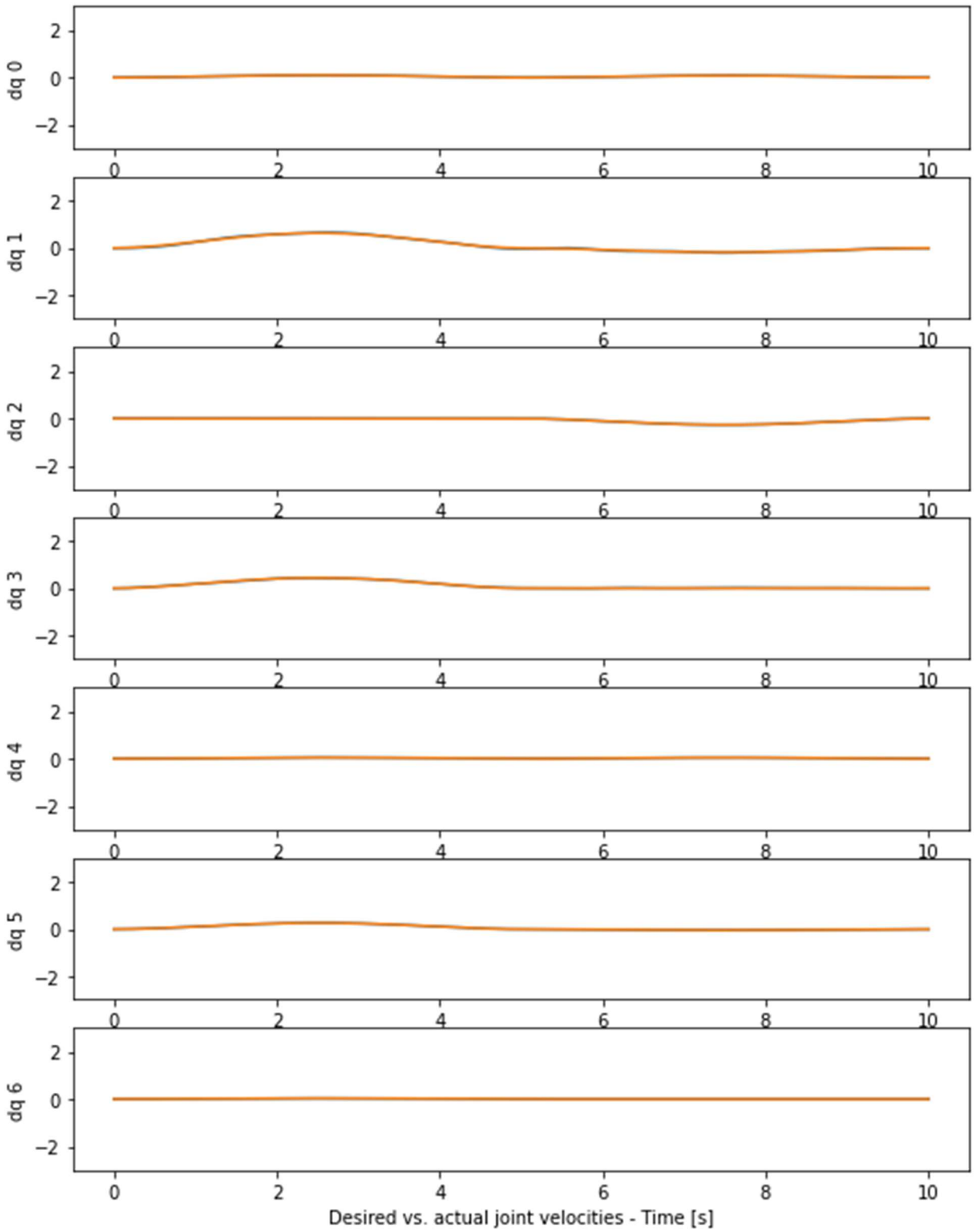
- Modified the robot controller to take desired velocity and position values from the equation above.
- End effector position



End effector velocity:

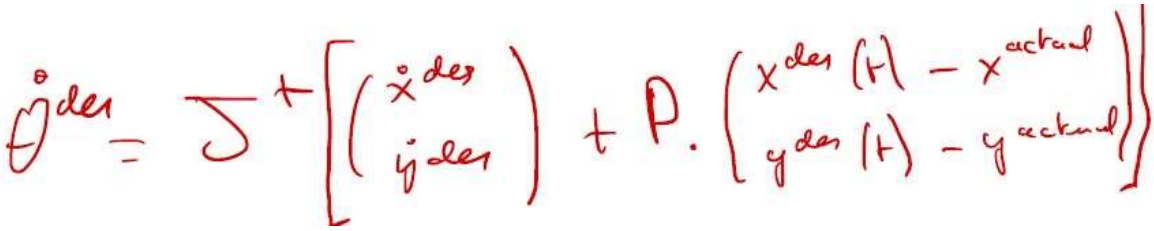



q 0
q 1
q 2
q 3
q 4
q 5
q 6
2.5
0.0
−2.5
0
2
4
6
8
10
Desired vs. actual joint positions - Time [s]

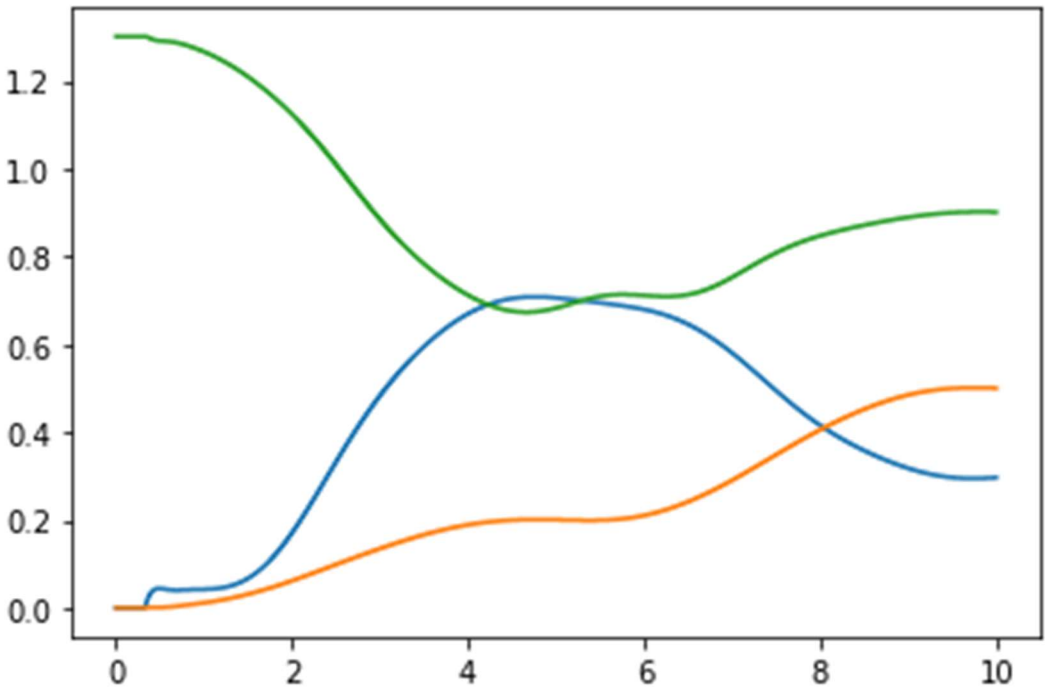

dq 0
dq 1
dq 2
dq 3
dq 4
dq 5
dq 6
2
0
−2
0
2
4
6
8
10
Desired vs. actual joint velocities - Time [s]

Question 3: End-effector control:

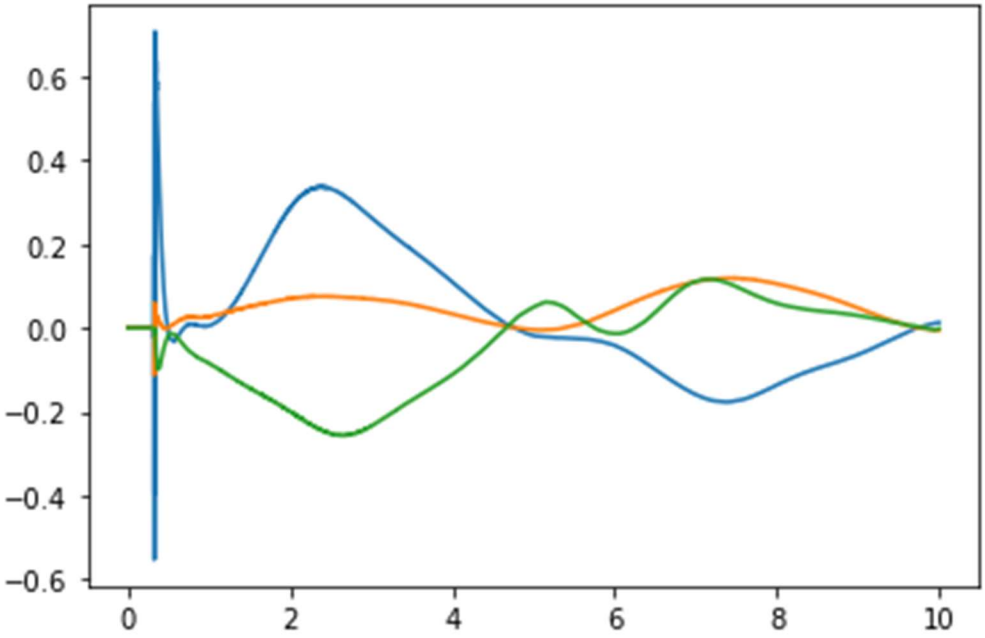
- Here we are using end effector position to control the movement of robot.
- We use the resolve rate control to find the desired joint velocities
- We calculate the Pseudo inverse using the joint positions and use the formula:



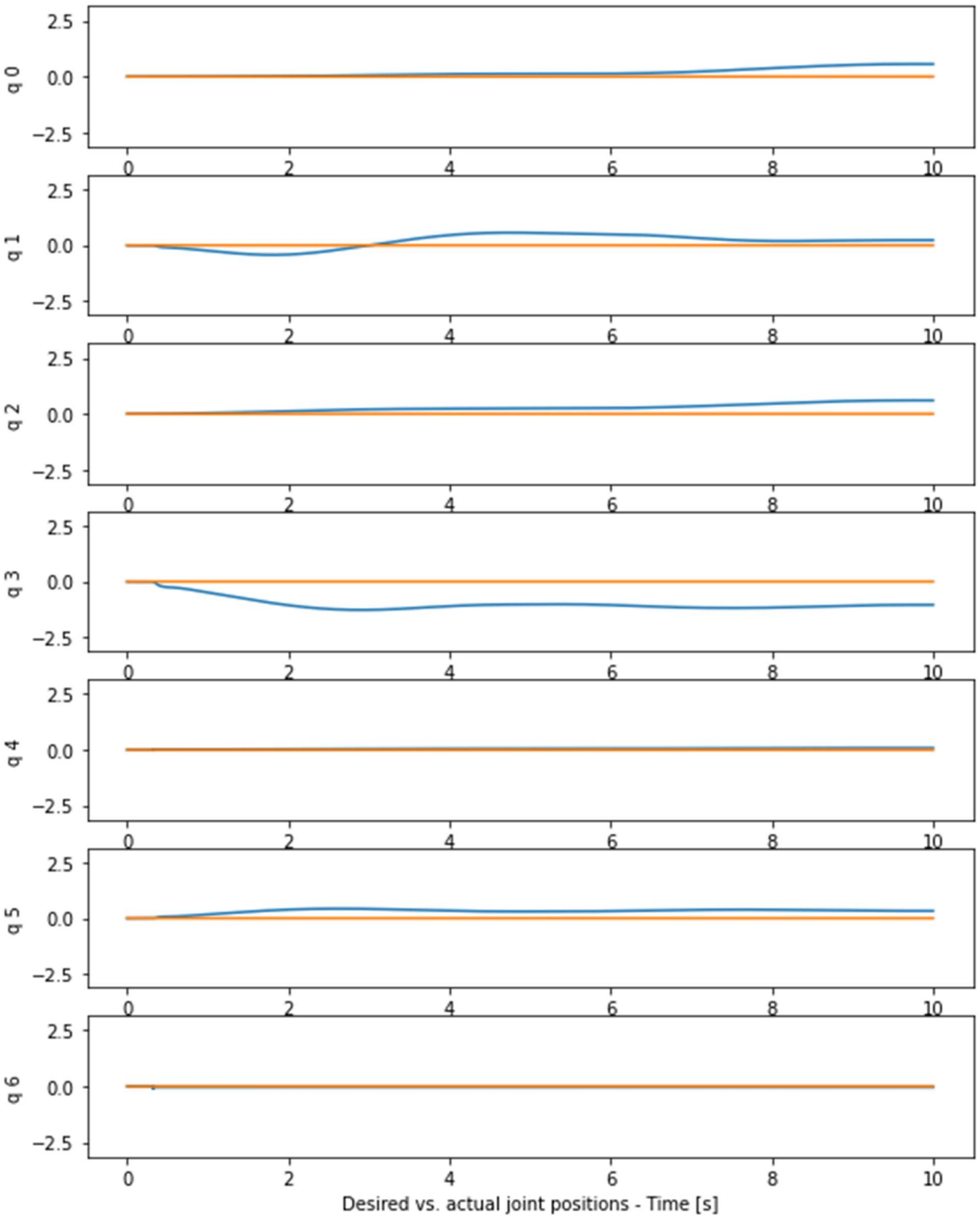
i.e. desired_joint_velocities = (J_sudo @ (desired_endeffector_velocities + np.diag(P) @ (desired_endeffector_positions - currp)))

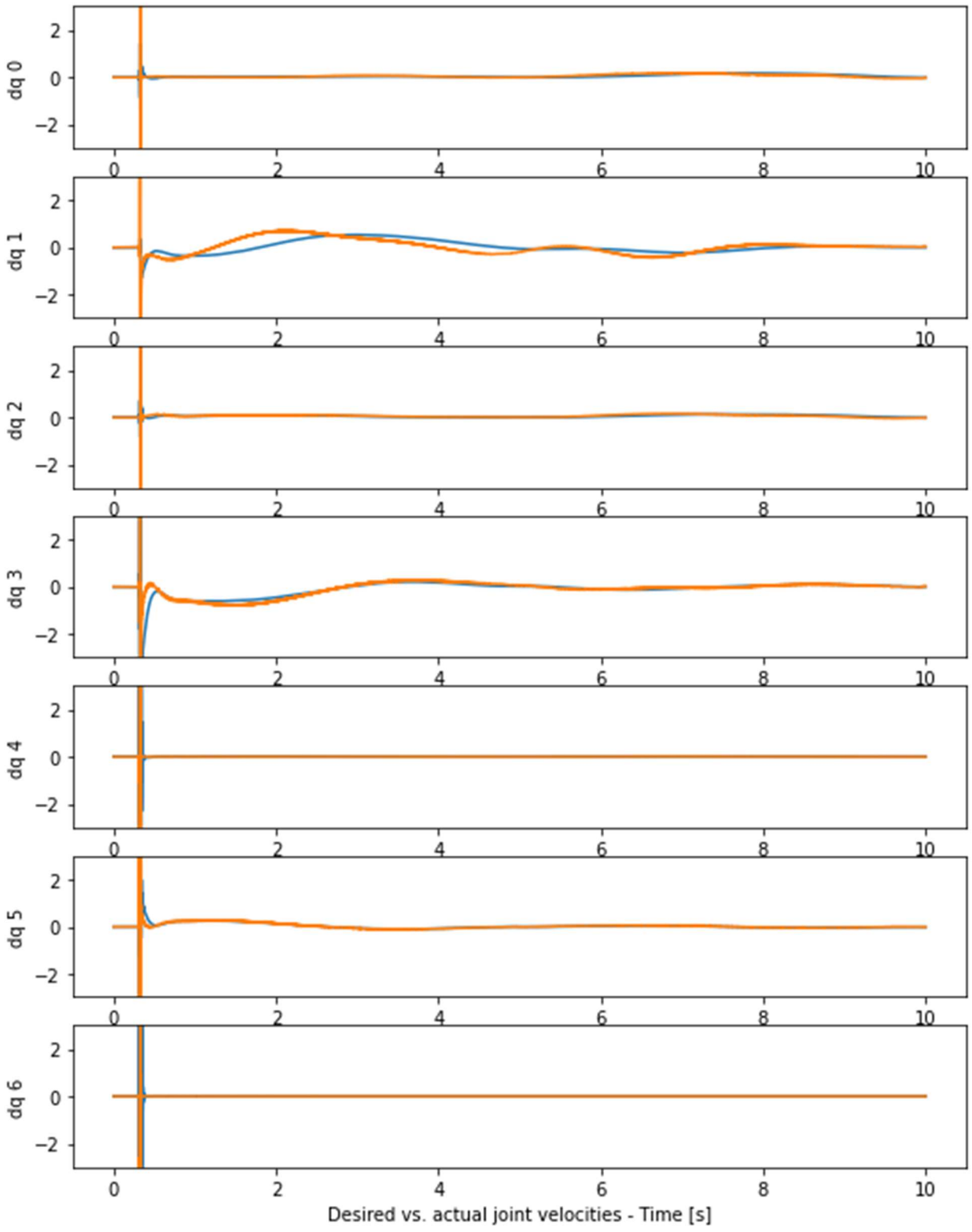


Position



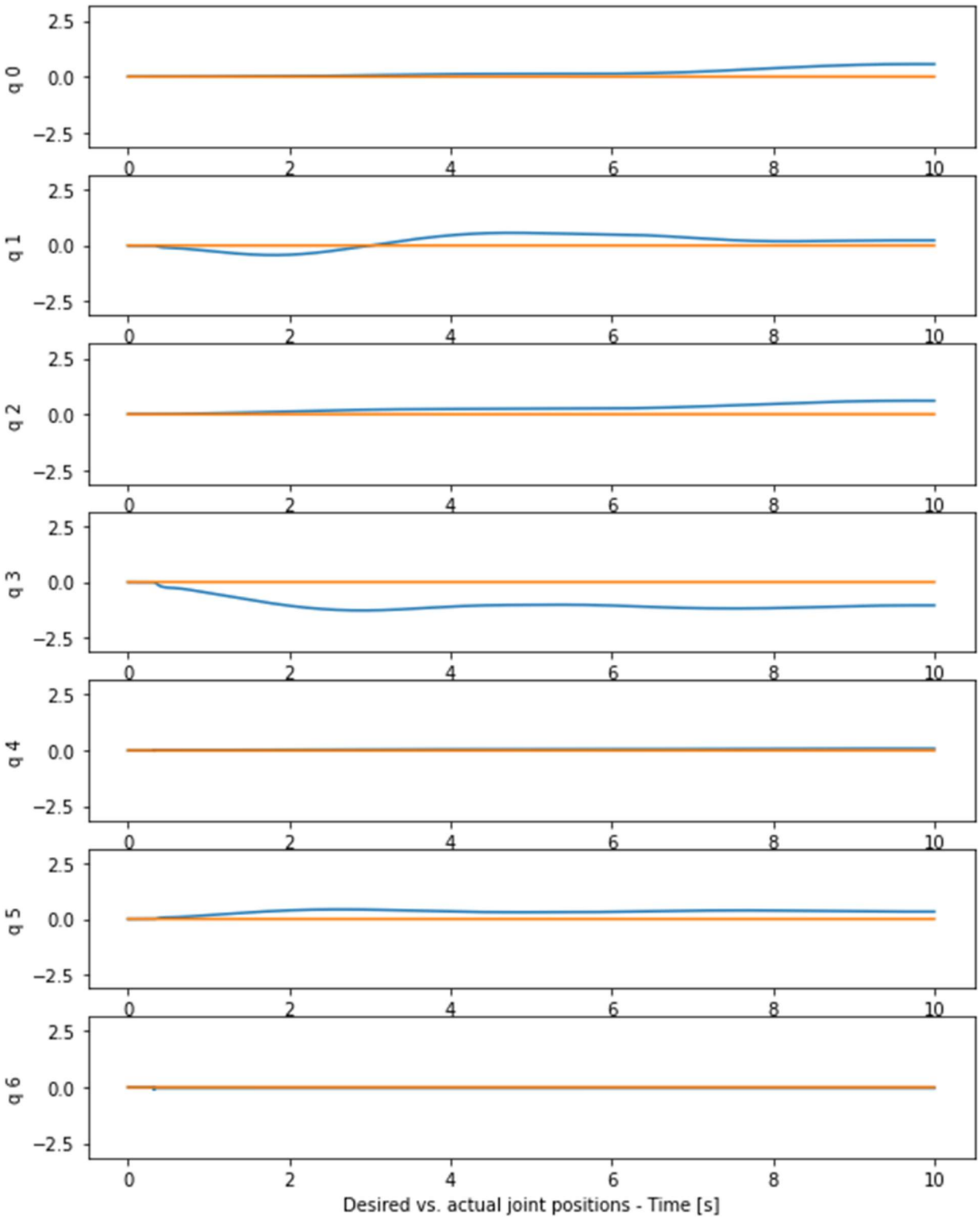
Velocity

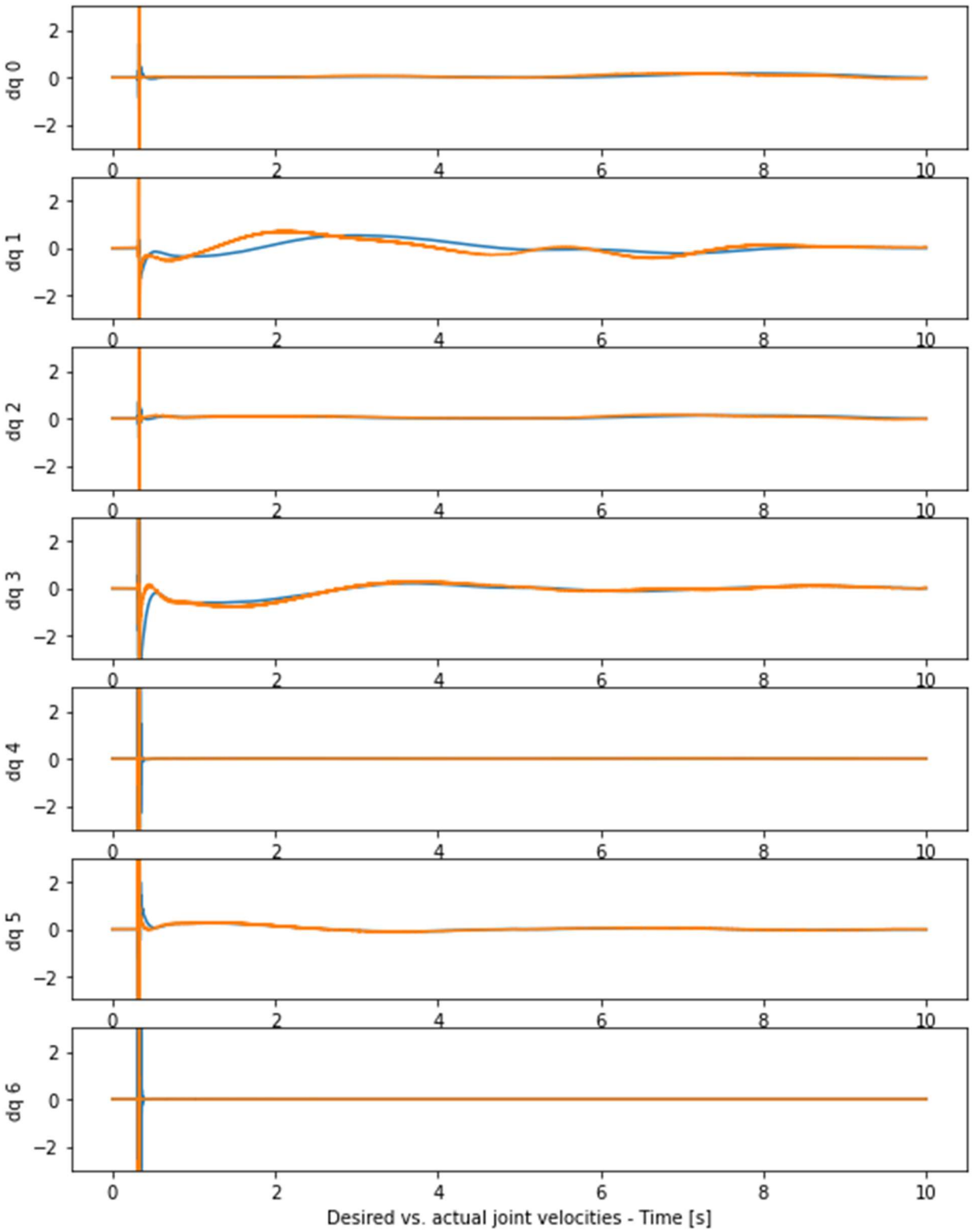

q 0
q 1
q 2
q 3
q 4
q 5
q 6
2.5
0.0
−2.5
0
2
4
6
8
10
Desired vs. actual joint positions - Time [s]

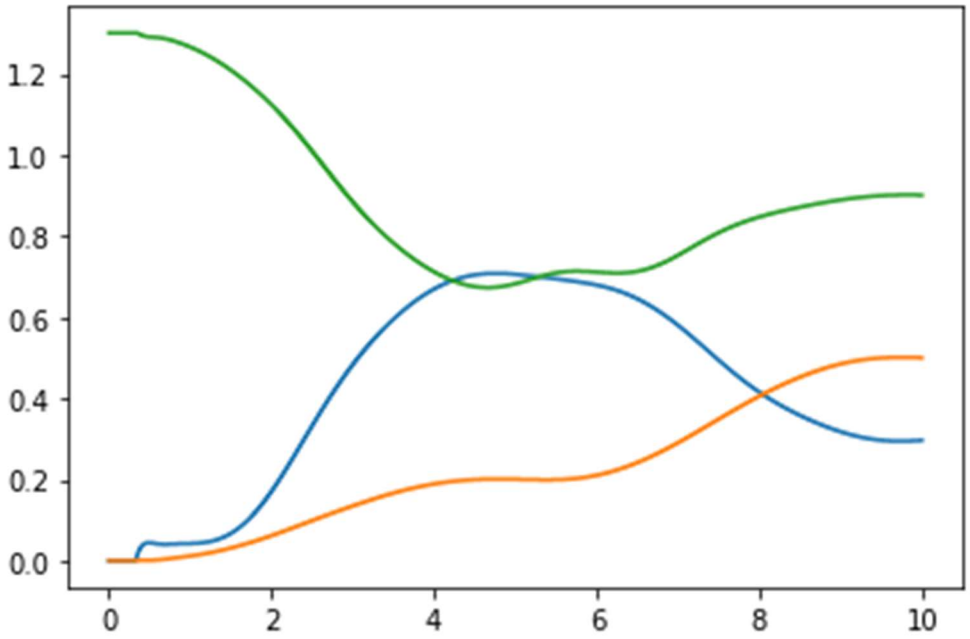

dq 0
dq 1
dq 2
dq 3
dq 4
dq 5
dq 6
2
0
−2
0
2
4
6
8
10
Desired vs. actual joint velocities - Time [s]

- We were able to manipulate the arm for both the desired positions, we found out the desired velocities with the constraints cause by the equation.
- There was a small overshoot for the first position when compared to question 2.

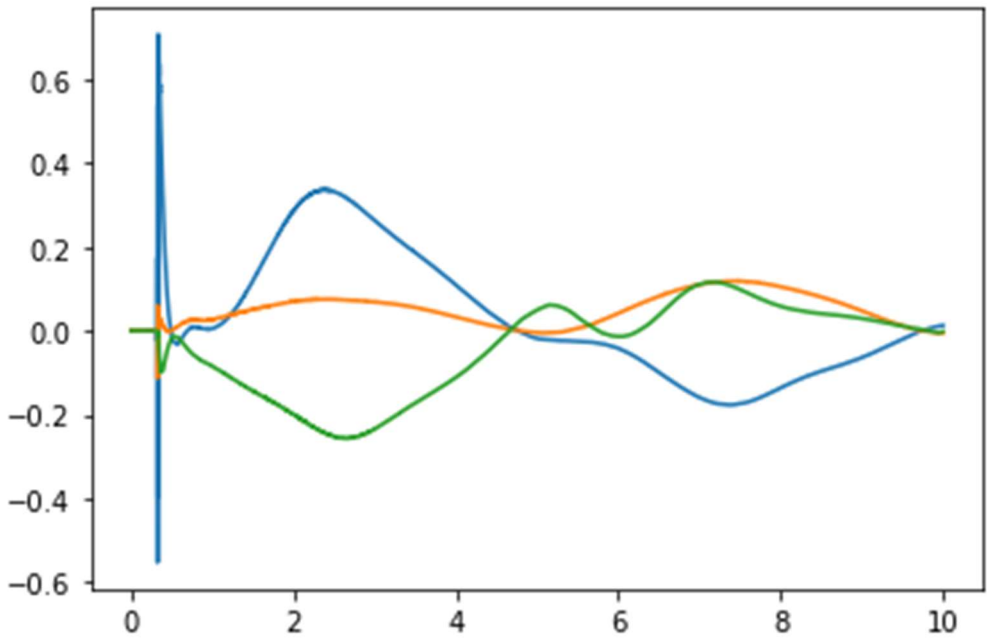
- **With Nullspace: [1,1,-1,-1,1,1,1] term, we see some different results:**


q 0
q 1
q 2
q 3
q 4
q 5
q 6
2.5
0.0
−2.5
0
2
4
6
8
10
Desired vs. actual joint positions - Time [s]


dq 0
dq 1
dq 2
dq 3
dq 4
dq 5
dq 6
2
0
−2
0
2
4
6
8
10
Desired vs. actual joint velocities - Time [s]



Position



Velocity

- We see different plots with nullspace in according to the desired theta values we put in.

Question 4: Impedance control and gravity compensation

We Know:

$$RNEA(\theta, \dot{\theta}, \ddot{\theta})$$

$$M\ddot{\theta} + C(\theta, \dot{\theta}) + G(\theta) = \tau$$

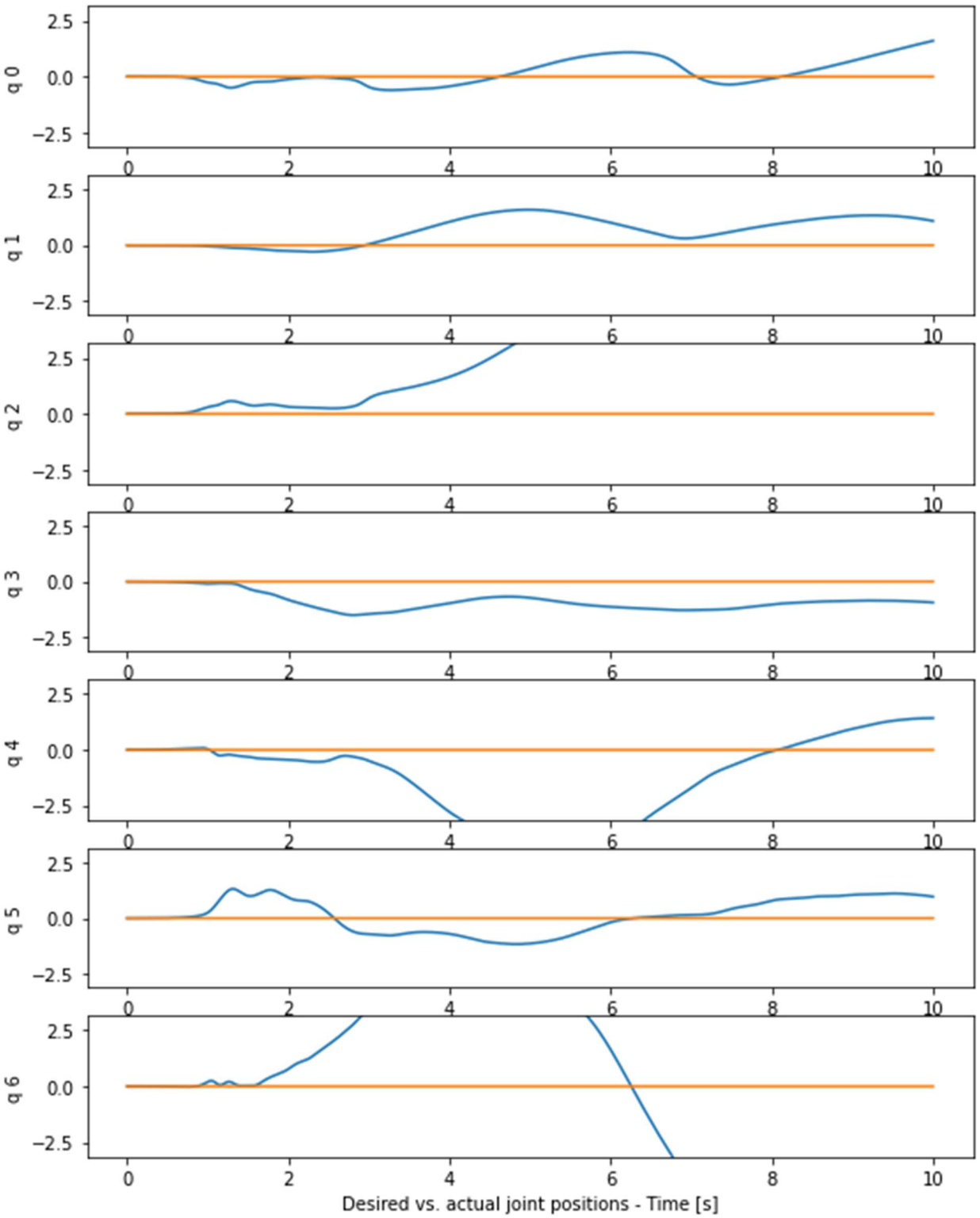
$$G(\theta) = RNEA(\theta, 0, 0)$$

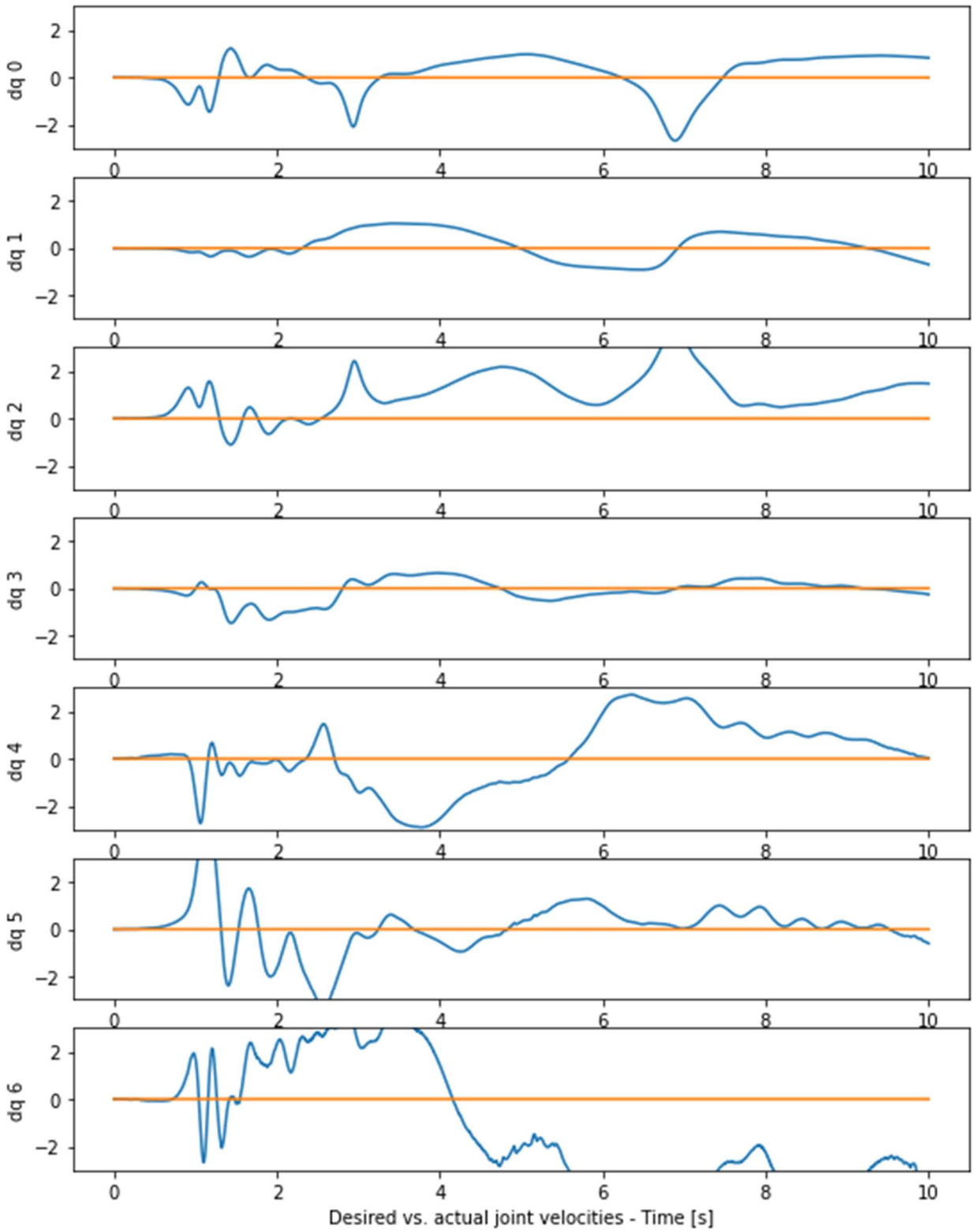
And we are given a RNEA function,

- We can find the gravity compensation term keeping the velocity and acceleration term 0.
- We can find the desired velocity by

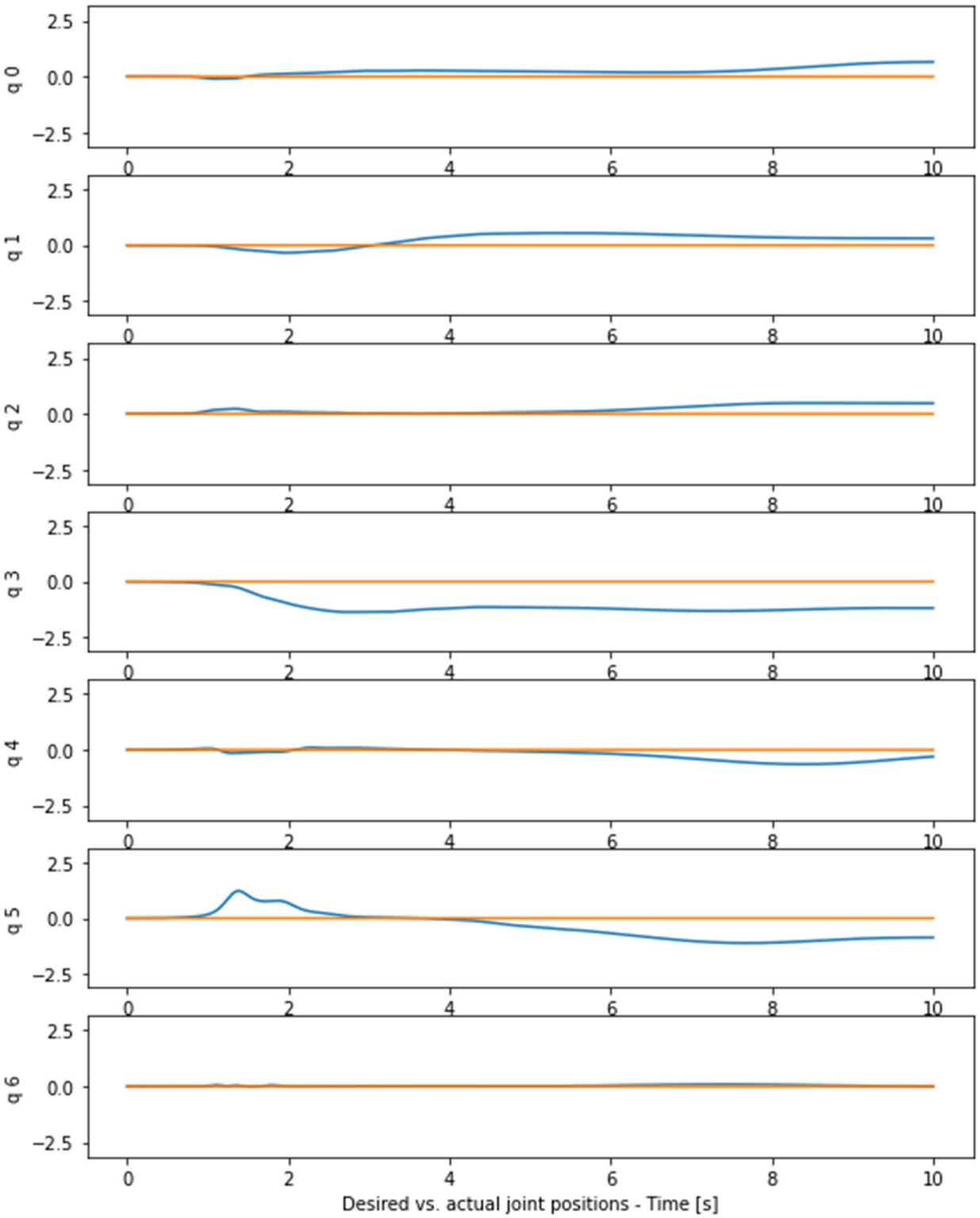
$$\tau = J^T\left(K(p_0(t) - p_{measured}) + D(\dot{p}_0(t) - \dot{p}_{measured})\right)$$

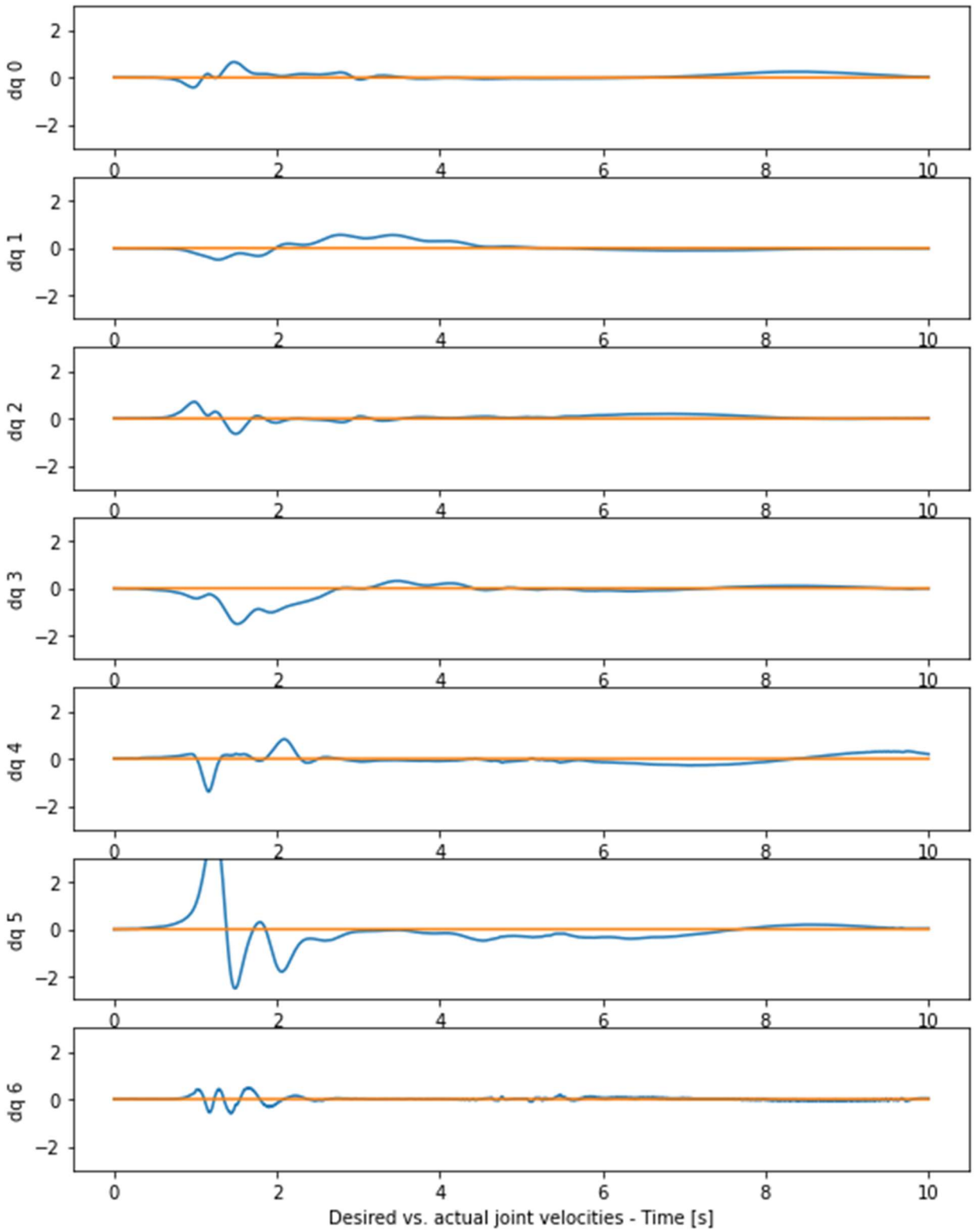
- We use forward kinematics and the equation to find the current velocity-position and observed velocity-position and use the equation above to find out the desired torque
- Plots:


q 0
q 1
q 2
q 3
q 4
q 5
q 6
2.5
0.0
−2.5
0
2
4
6
8
10
Desired vs. actual joint positions - Time [s]


dq 0
dq 1
dq 2
dq 3
dq 4
dq 5
dq 6
2
0
−2
0
2
4
6
8
10
Desired vs. actual joint velocities - Time [s]

With the damping factor We found that the controller wa more stable.
Plots:


q 0
q 1
q 2
q 3
q 4
q 5
q 6
2.5
0.0
−2.5
0
2
4
6
8
10
Desired vs. actual joint positions - Time [s]


dq 0
dq 1
dq 2
dq 3
dq 4
dq 5
dq 6
2
0
−2
0
2
4
6
8
10
Desired vs. actual joint velocities - Time [s]